Survey of Meta-Heuristic Algorithms for Deep Learning Training

Zhonghuan Tian and Simon Fong

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/63785

Abstract

Deep learning (DL) is a type of machine learning that mimics the thinking patterns of a human brain to learn the new abstract features automatically by deep and hierarchical layers. DL is implemented by deep neural network (DNN) which has multi-hidden layers. DNN is developed from traditional artificial neural network (ANN). However, in the training process of DL, it has certain inefficiency due to very long training time required. Meta-heuristic aims to find good or near-optimal solutions at a reasonable computational cost. In this article, meta-heuristic algorithms are reviewed, such as genetic algorithm (GA) and particle swarm optimization (PSO), for traditional neural network's training and parameter optimization. Thereafter the possibilities of applying meta-heuristic algorithms on DL training and parameter optimization are discussed.

Keywords: deep learning, meta-heuristic algorithm, neural network training, nature-inspired computing algorithms, algorithm design

1. Introduction

Deep learning (DL) is a branch of machine learning. Based on a set of algorithms, DL attempts to model high-level abstractions in data by using multiple processing layers with complex structures. Developed by Professor Hinton [1] in 2006, DL now is becoming the most prevalent research area in machine learning. DL is a collective concept of a series of algorithms and models including but not limited to convolutional neural networks (CNNs), deep belief networks (DBNs), restricted Boltzmann machines (RBMs), deep Boltzmann machines (DBM), recursive auto encoders and deep representation respectively, regarding the problem to be solved. The most famous models for machine learning are RBM and DBN and CNN for image classification.

DL is the so-called "Deep" towards those widely used "shallow learning" algorithms such as support vector machine (SVM), boosting and maximum entropy method. Those shallow learning extracts feature mostly by artificial sampling or empirical sampling, thus the model or network will learn a non layer structure feature. On the contrary, DL learns raw data layer by layer by transforming data from raw feature space to transformed feature space. Additionally, deep structure can learn and approach nonlinear function. All these advantages are beneficial to classification and feature visualization [2, 25].

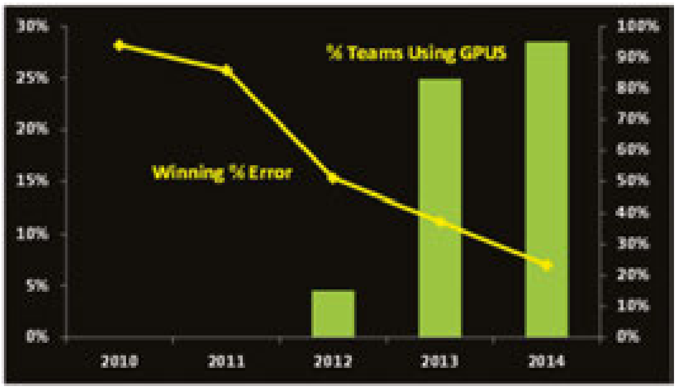

Figure 1. Error rate of Image Net competition.

To process big data and large scale dataset, DL has been widely used in many research areas and industrial areas including pattern recognition, image classification and natural language processing (NLP). For instance, in Image Net large-scale visual recognition challenge (ILSVRC) [3], DL and convolutional neural network are first implemented in image classification and made a great success. As shown in **Figure 1**, the error rate declines from 25 to 15% when DL and CNN are implemented. From then on, "DL + big data + GPU" is integrated into traditional image classification application, and companies such as Google and Baidu have successfully updated their image searching production by DL's implementation [4].

Though DL outperforms those shallow learning methods and it has been widely adopted by the R&D industry, its application and implementation still have some shortcomings. A large deep neural network (DNN) may have millions of parameters and is mostly trained by contrastive divergence (CD) learning algorithm which is iterative and known to be time consuming [5]. The most significant problem is that when facing very large scaled data the DNN will take several days or even months to learn even though the greedy search strategy is proposed. In this case, many companies and researchers are trying to improve hardware capabilities and applying high-processing facilities such as GPU and parallel computing. Others are focusing on using other training algorithms to substitute traditional CD to speed up the training process.

Meta-heuristic algorithms aim to find global or near-optimal solutions at a reasonable computational cost. In terms of meta-heuristic, the words of "meta" and "heuristic" are Greek, where "meta" is "higher level" or "beyond" and heuristics means "to find", "to know", "to guide an investigation" or "to discover". Heuristics are methods to find good (near-) optimal solutions in a reasonable computational cost without guaranteeing feasibility or optimality. In other words, meta-heuristics are a set of intelligent strategies to enhance the efficiency of heuristic procedures [6].

Meta-heuristic algorithms usually rely on different agents such as chromosome (genetic algorithm [GA]), particles (particle swarm optimization [PSO]), and firefly (firefly algorithm), searching iteratively to find the global optimum or near-global optimum. Many strategies such as evolutionary strategy, social behavior and information transition are implemented to guarantee the whole population will move towards global optimum iteratively and protect from falling in local optimum.

Traditional artificial neural network (ANN) used to have multilayer feed-forward neural network structure and use back-propagation (BP) to modify the weight. The mainly used strategy is gradient descent (GD). Meta-heuristic has been successfully implemented on traditional neural network to speed up the training process by substituting the GD strategy with iterative evolutionary strategy or swarm intelligence strategy [5, 7–10].

Gudise and Venayagamoorthy [5] compared feed forward with PSO and feed forward with BP, and the results show that feed forward with PSO is better than BP in terms of nonlinear function.

Leung et al. [10] presents the tuning of the structure and parameters of a neural network using an improved GA. It will also be shown that the improved GA performs better than the standard GA based on some benchmark test functions.

Juang [8] proposed a new evolutionary learning algorithm based on a hybrid of GA and PSO called HGAPSO. In each epoch, the upper-half of the GA population are defined as elites and the rest will be discarded. Enhanced by PSO and GA, the elites will form the next generation of GA. The hybrid method outperforms PSO and GA in recurrent or fuzzy neural network.

Meissner et al. [9] used optimized particle swarm optimization (OPSO) to accelerate the training process of neural network. The main idea of OPSO is to optimize the free parameters of the PSO by having swarms within a swarm. Applying the OPSO method to neural network training with the aim to build a quantitative model, OPSO approach yields parameter combinations improving overall optimization performance.

Zhang et al. [7] proposed a hybrid algorithm of PSO plus BP for neural network training. By utilizing the advantage of PSO's global searching ability as well as BP's deep search ability, the hybrid algorithm has a very good performance in both convergent speed and convergent accuracy.

Structures of DL model are similar to the traditional ANN, some modifications are implemented for better learning ability. For instance, the CNN is a traditional ANN modified with pooling procession and the structure of RBM is an undirected graph or a bidirectional neural network. DL model shares similar model with neural network and may take different training algorithm instead of GD strategy.

In this survey, the relevant work of applying meta-heuristic algorithms for ANN's training is reviewed. In addition, the structure and working process of RBM are analyzed and as the basic structure of DL model, RBM's training process is introduced. We survey the possibility of implementing meta-heuristic algorithms on RBM's parameter training process.

The rest of the sections is organized as follows: meta-heuristic algorithms including GA and PSO are introduced in Section 2. ANN is introduced in Section 3. The implementation of meta-heuristic on ANN is introduced in Section 4. DL and RBM are introduced in Section 5. Some conclusion and discussion are drawn in Section 6.

2. Meta-heuristic algorithm

Meta-heuristic is a collective concept of a series of algorithms including evolutionary algorithm, the most famous one is GA [11], naturally inspired algorithm, the most prevalent one is PSO [12], trajectory algorithm, such as Tabu search [13], and so on. The classification of meta-heuristic is shown in **Figure 2**. In this paper, I mainly focus on surveying GA and PSO because these two are the most prevalent as well as most widely implemented. GA and its working process are introduced in Section 2.1 and PSO is introduced in Section 2.2, respectively.

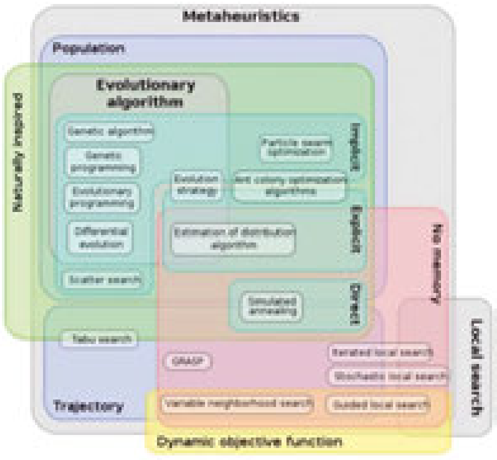

Figure 2. Classification of meta-heuristic algorithm.

2.1. Genetic algorithm

Founded in 1975 by Professor Holland, GA sets up an evolution model that simulates Darwinian genetic selection and the natural elimination process [11]. Chromosomes carry information and the operation of crossover and mutation on chromosome guarantees the whole algorithm could find the global optimum or near-global optimum iteratively. Crossover operation keeps the whole population moving towards the global optimum through giving better chromosome higher chance for propagation. Mutation operation keeps the whole population's diversity and avoids population falling in local optimum easily. The main working process could be summarized as follows:

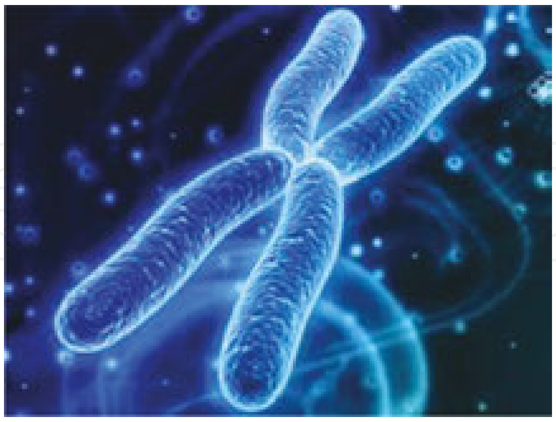

Figure 3. Chromosome image in biology.

Step 1. Chromosomes which carry information are randomly generated.

Step 2. All chromosomes are evaluated with fitness functions to calculate fitness values of each as $fitness_i$.

Step 3. Based on Russian roulette strategy, chromosomes are randomly chosen as parents for next generation.

Step 4. Parents do crossover operations to get next generations.

Step 5. Do mutation operation on each chromosome.

Step 6. Back to Step 2 until meeting the stop criteria or exceeding pre-set iteration number.

A biological picture of chromosome is shown in **Figure 3**. The working flow of GA is shown in **Figure 4**.

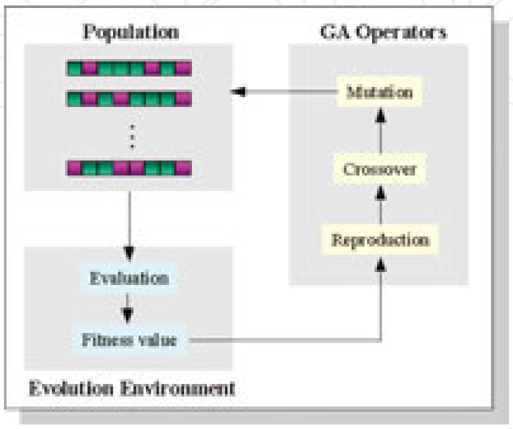

Figure 4. GA working flow.

2.1.1. Russian roulette strategy

Russian roulette strategy is designed for choosing the parents of the next generation, as shown in **Figure 5**. Each chromosome has a possibility p_i of being chosen as parents.

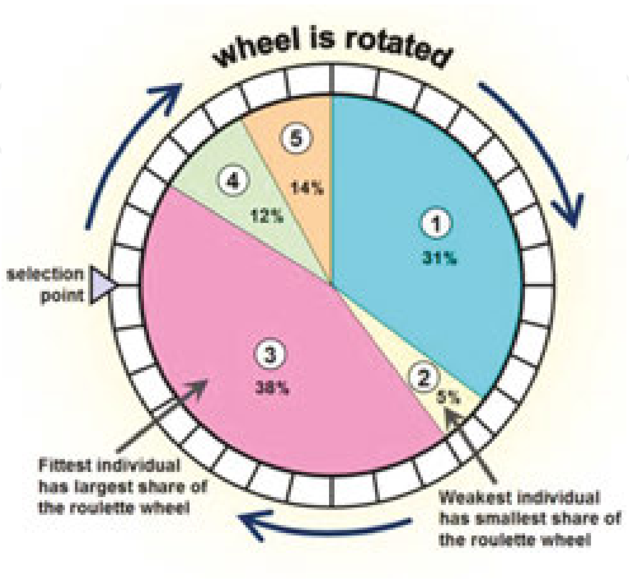
wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 5. Russian roulette strategy.

p_i is calculated as below for each chromosome i:

$$p_i = \frac{fitness_i}{\sum_{i=1}^{n} fitness_i}$$

This strategy keeps the idea that the chromosome with higher fitness value will have higher possibility of being chosen thus make sure better chromosome's information will not be missed or has few possibilities of missing in the next generation. In the meantime, it also allows those chromosomes without better fitness value to transmit their information to the next generation [11].

2.1.2. Crossover and mutation

Figure 6 shows us a typical crossover operation and mutation operation. The most common way of applying crossover operation is selecting a site, cutting up each chromosome and reconstructing both to get next generation. If their fitness value is better than their parents, submit as next generation otherwise keep their parents as next generation. Furthermore, regarding different real problem to be solved, different crossover operation such as multisite crossover would be used and these will be discussed later.

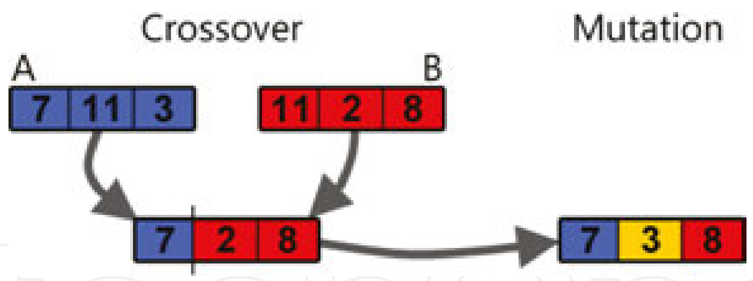

Figure 6. Crossover and mutation.

Figure 6 also shows us the typical mutation operation in GA. The most common way of applying mutation is to randomly select some site of chromosome and randomly change it with any possible value regardless of the fitness value will be better or not. Of course the mutation operation may make one chromosome "worse" while mutation operation can keep the population's diversity and avoid falling in local optimum.

2.1.3. Fitness function and genetic coding strategy

GA is first used to optimize complex continuous functions which are not able to calculate its mathematical global optimal location. Thus the design of chromosome itself, fitness function and crossover operation are very easy. However, when trying implementing GA on real problems, many types of problems such as discrete problem, time series problem and multi objective problem are faced. In this way, how to design the chromosome itself to represent the real problem better and how to calculate its fitness value when given chromosome need reconsidering. To solve the above troubles, genetic coding strategy is put forward which encodes real problem to chromosome and calculates its fitness value. When whole optimizing is finished, the best chromosome will be decoded to real problem's expression and then analyzed and discussed why it is global best.

2.2. Particle swarm optimization

PSO is developed by Professor James Kennedy and Russell Eberhart [12]. The main difference of PSO is that it takes "social behaviors" and "memory" to guarantee whole population's moving towards global optimum iteratively instead of evolutionary strategy.

PSO optimizes a problem by having a population of candidate solutions and moving these particles around in the search-space according to simple mathematical formula over the particle's position x_i and velocity v_i. Each particle has memory and its movement is influenced by its local best known position $pbest_i$ but is also guided toward the best known positions in the search-space, which are updated as better positions found by all particles called *gbest*.

The iterative formulas of PSO are

$$v_{i+1} = v_i * w + c_1 * rand * \left(pbest_i - x_i\right) + c_2 * rand * \left(gbest - x_i\right)$$

$$\textit{Velocity constraint} \; ()$$

$$x_{i+1} = x_i + v_{i+1}$$

$$\textit{Search range constraint} \; ()$$

w is called the inertia weight that controls the exploration and exploitation of the search space because it dynamically adjusts velocity and is usually set as 0.8. c_1 and c_2 are called learning factors and usually set as 2.

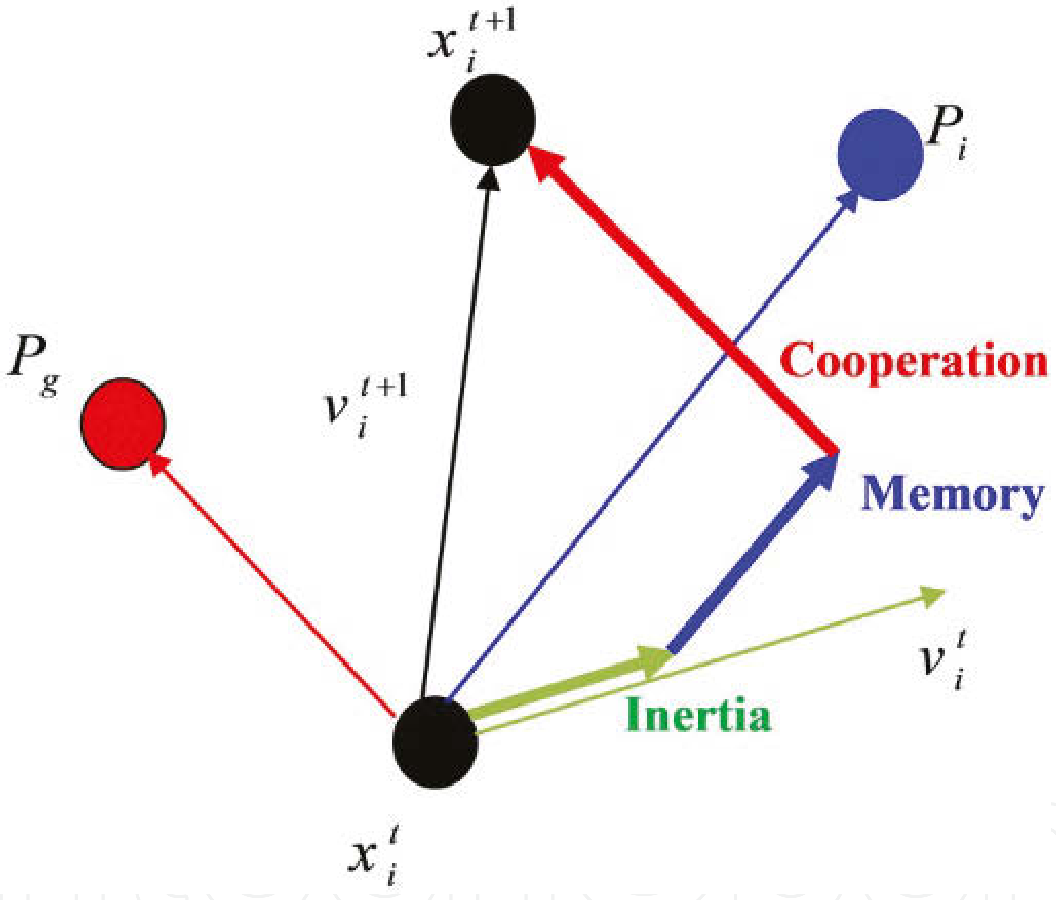

Figure 7. Schematic of PSO.

In the iterative formula of PSO, $v_i * w$ refers to the particle keeps its own search direction. $c_1 * rand * (pbest_i - x_i)$ refers to the particle's movement is influenced by its own personal best record. $c_2 * rand * (gbest - x_i)$ refers to the particle's movement is influenced by whole population's best record. To keep whole population will not premature or overfit, when implementing iteration process, a limitation of v_{max} *and* v_{min} is predefined to constraint particles' movement. Additionally, the search space is also predefined thus for each particle's location x_i, there are upper bound x_{max} and lower bound x_{min} to make sure each particle will not be out of range. A schematic diagram is shown in **Figure 7** and the working flow of PSO is shown in **Figure 8**.

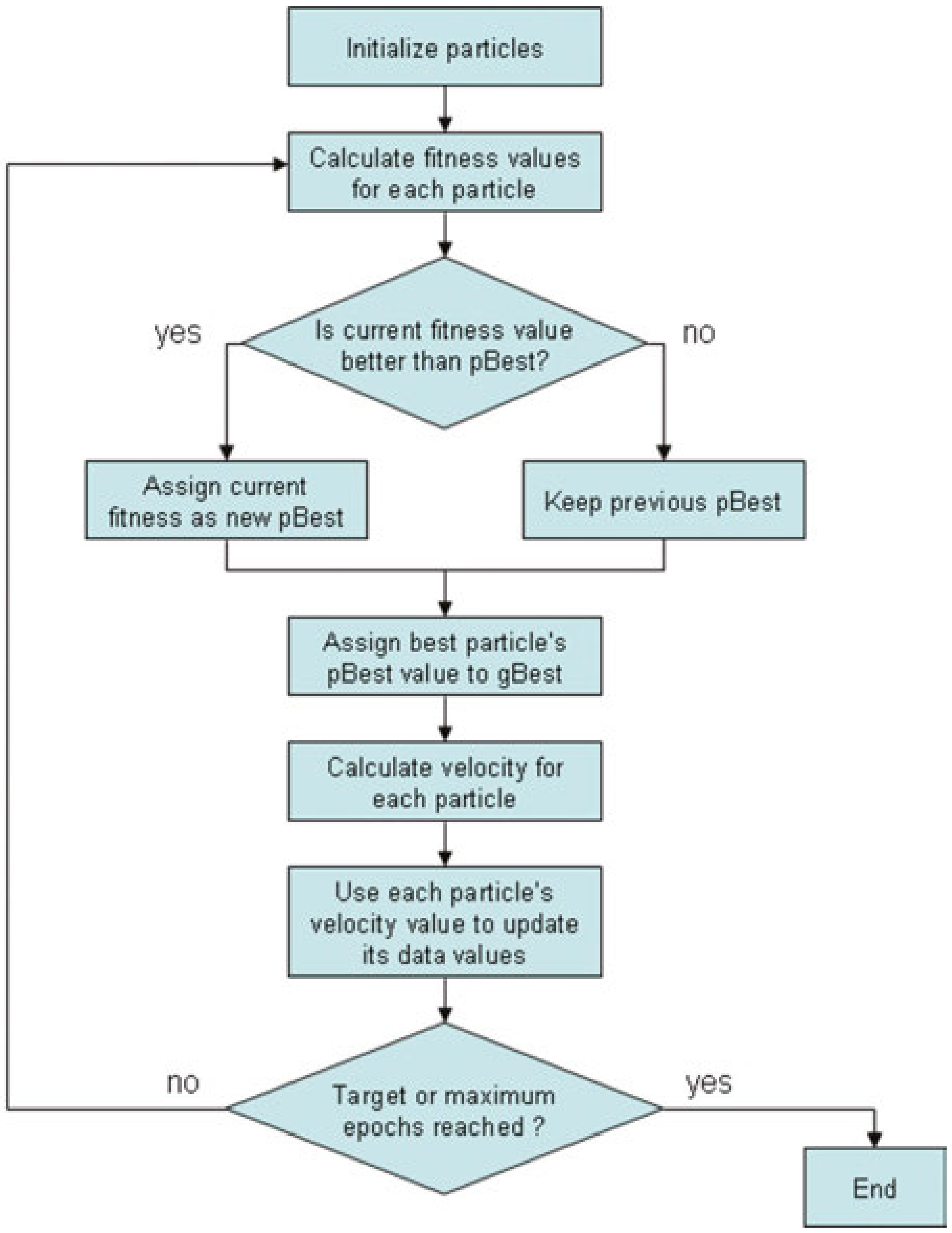
Initialize particles
Calculate fitness values for each particle
Is current fitness value better than pBest?
yes
no
Assign current fitness as new pBest
Keep previous pBest
Assign best particle's pBest value to gBest
Calculate velocity for each particle
Use each particle's velocity value to update its data values
Target or maximum epochs reached ?
no
yes
End

Figure 8. Working flow of PSO.

PSO is prevalent because of its concise evolutionary strategy as well as its concise operations. Unlike GA's crossover and mutation operation and complex genetic coding strategy, PSO only needs randomly generated location x and velocity v. However, naïve PSO is designed for math function optimization which is a continuous problem. For application, researchers still need to design own coding strategy to make sure PSO's iterative formulas are meaningful, especially for discrete problems. One example is applying PSO on graph optimization to solve some NP-hard problem such as travelling salesman problem (TSP) problem [14]. TSP asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

With the development of meta-heuristic, researchers are beginning to try implementing meta-heuristic on TSP in case to get a "good enough" result in a reasonable computation time. To solve and adapt TSP to PSO, researchers innovate a concept of "swap operator" and "swap sequence" which gives the "+" operator a new meaning [14, 15].

3. Neural network

In machine learning and cognitive science, ANNs are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. ANNs are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning [16]. The ANN was developed in the 1950s, there are almost three generations of neural networks: perceptron [17], feed-forward BP neural network [18] and spiking neural network (SNN) [19].

3.1. Perceptron

Founded in the 1950s by Professor Rosenblatt, the basic perceptron is put forward as a probabilistic model to simulate human brain's working process when receiving information. **Figure 9** shows us a typical structure of perceptron that only has one layer, with multi-input neurons as (t_1, t_2, $\cdots$, t_n) in which k is the total number of neurons. Their corresponding inputs are (x_1, x_2, $\cdots$, x_n). Each neuron has its own weight (w_{1j}, w_{2j}, $\cdots$, w_{nj}) in which j refers to the output layer. The net input $net_j = \sum_{n}^{i=1} x_i * w_{ij}$ is processed by the activation φ. If $\varphi(net_j) > threshold\ \theta_j$, then the output neuron is activated and output $o_j = 1$, otherwise output neuron is not activated and output $o_j = 0$. The training process of perceptron is that suppose the desired output is o_d and actual output is o_a, then

$$\Delta w_{ij} = \left(o_d - o_a\right) * x_i$$

$$w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}$$

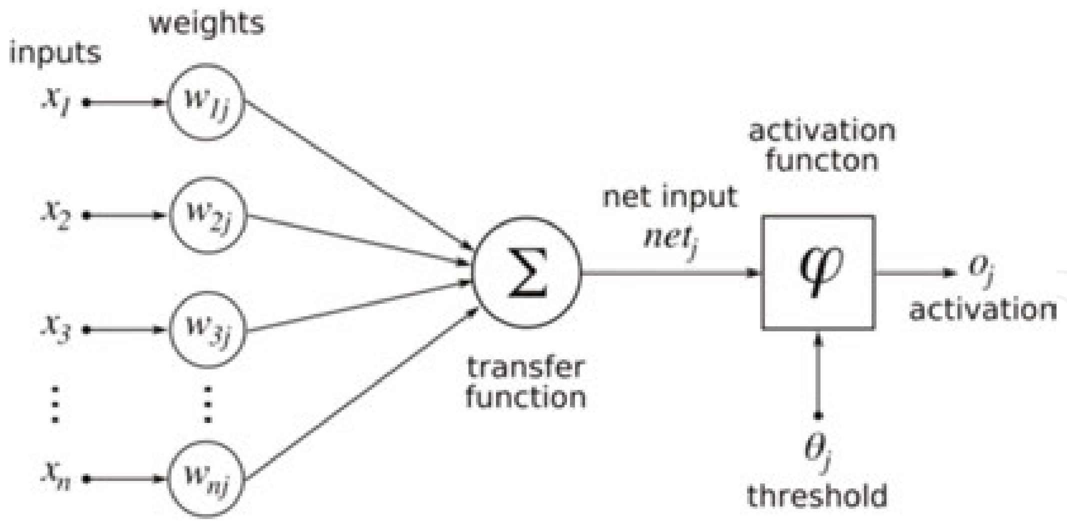

Figure 9. Structure of perceptron.

By using these training strategies, when $|(o_d - o_a)| < eps$, in which eps is a predefined accuracy, i.e., 10^{-5}, the perceptron could be seen as mature. This training strategy is the ancestor of the widely used BP algorithm.

Geometrically, trained perceptron could be seen as a linear function. It can correctly classify the linear separable problem but not linear inseparable problem such as XOR problem (**Figure 10**). XOR problem contains two classes A and B while the diagonal belongs to the same class, i.e., data (0, 0) and data (1, 1) belong to class A while data (1, 0) and (0, 1) belong to class B. This will cause no lines or linear functions can separate these two categories correctly unless using curves or reflect the data to high-dimensional space [18].

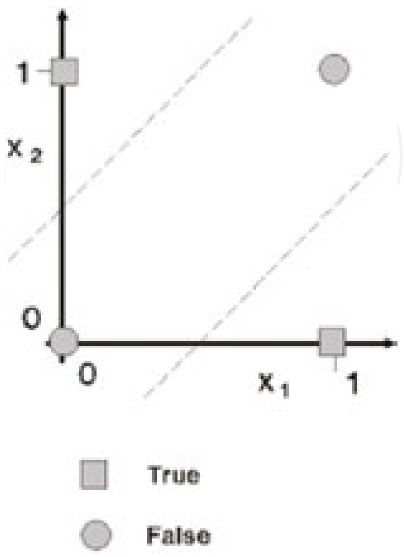

Figure 10. XOR problem.

3.2. Feed-forward neural network

Figure 11 shows us a typical multilayer unidirectional feed-forward neural network and **Figure 12** shows us a typical recurrent neural network, Hopfield network. The most prevalent structure of three-layer neural network with input layer, hidden layer and output layer, every two neurons are connected with weight *w*.

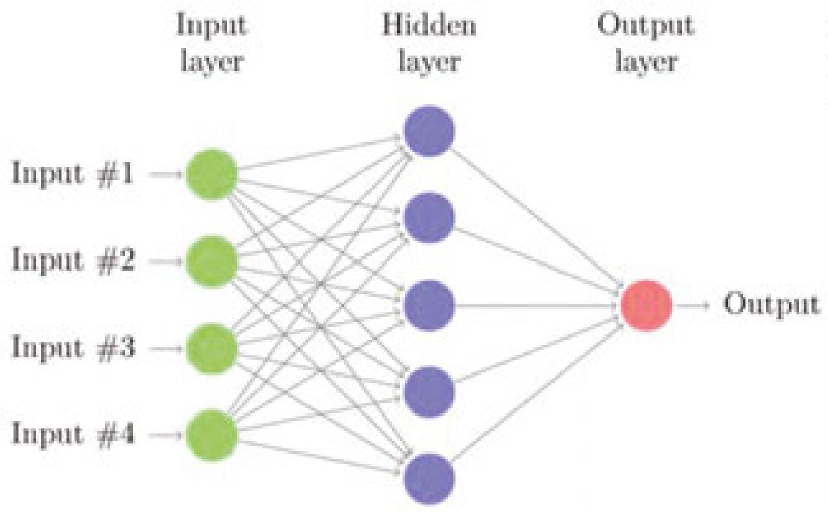

Figure 11. Feed-forward neural network.

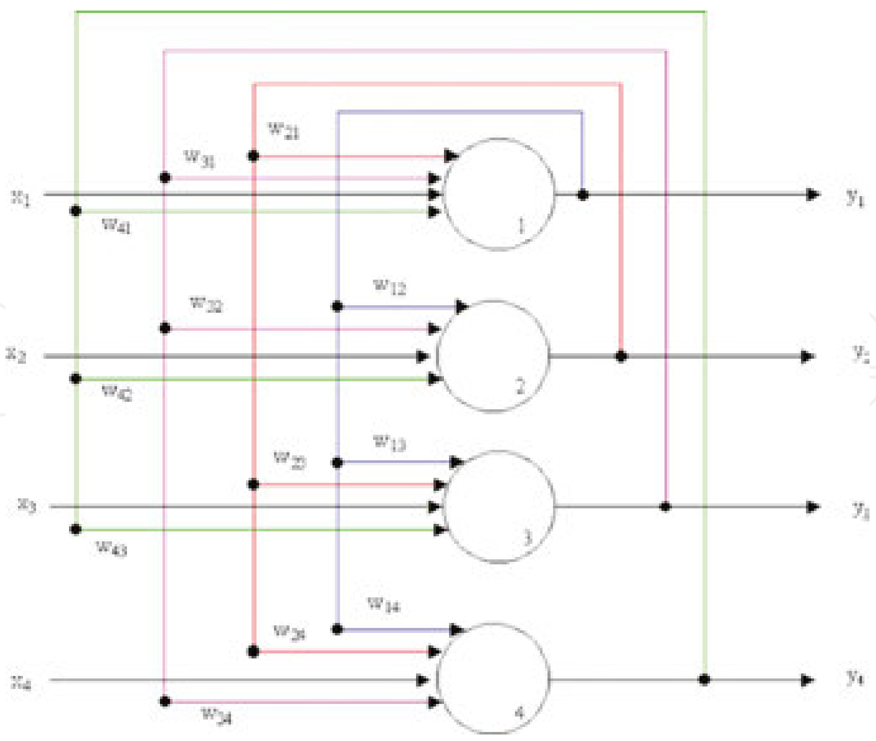

Figure 12. Hopfield neural network.

In a typical feed-forward neural network, the information is transmitted forward from input layer to hidden layer and then from hidden layer to output layer. It works similar to perceptron above, for each neuron *j* in hidden layer, calculate

$$y_j = \sum_{i=1}^{n} x_i * w_{ij}$$

Then for each neuron *h* in output layer, calculate

$$o_h = \sum_{i=1}^{n} y_i * w_{jh}$$

The output layer could have one or more neurons to represent the final result, different output layer structure may lead to different training time and prediction accuracy.

BP algorithm is the most prevalent supervised learning algorithm for neural network. BP's main idea is using GD algorithm to find the gradient-descent-most way to modify the weight in neural network. The modification starts from the output layer, then to hidden layer and input layer. Because the GD is based on the difference or error between the desired output and actual output, BP is also called error-BP method. Suppose desired output is o_d and actual output is o_a, the error function is defined as:

$$\mathrm{E} = \frac{1}{2}(o_d - o_a)^2$$

Suppose the layers are input layer *i*, hidden layer *j* and output layer *h*, respectively, using partial derivative,

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}}, \Delta w_{jh} = \eta \frac{\partial E}{\partial w_{jh}}$$

in which η is called learning rate, usually set as 0.01 or 0.1 according to the set of weight of the network. Weight *w* is usually set as a random number in [0, 1] in case the learning process is too long. Then, update the weight between each neurons,

$$w_{ih} = w_{ih} + \Delta w_{ih}, w_{hj} = w_{hj} + \Delta w_{hj}$$

One epoch of training process is finished. GD will modify weight iteratively using the formula above till the whole network is trained mature.

Unlike perceptron, multilayer network can represent complex nonlinear function with hidden layers, thus can process XOR problems well. To some extent, multilayer network can represent any type of nonlinear function by using hidden layers. However, when implementing BP, if the depth of network is over 5, the error transmitted back to input layer or first hidden layer will be decayed significantly to almost 0 thus making the modification useless, so in real application, the depth of network is smaller than 5 [16].

3.3. Spiking neural network

SNN is recognized as the third generation of neural network. Maass [19] proved the spiking neuron and SNN has a powerful learning ability and information processing ability than traditional neural network. Put forward by Gerstner [20] in 1997, the SNN gives us a concise biological neuron model processed by temporal coding called spiking neuron model (SRM).

Figure 13(A) and **(B)** shows us the basic structure of SNN, the network structure and information transition of neural network, SNN is similar to using feed-forward multilayer neural network. However, the transition process is a little different with feed-forward neural network. Between neurons *i* and *j*, there are multiple delays (d_1, d_2, $\cdots$, d_k) and every link has its own weight (w_{ij}^1, w_{ij}^2, $\cdots$, w_{ij}^k) corresponding to each delay. Compared to traditional NN, SNN has more powerful learning ability and information processing ability.

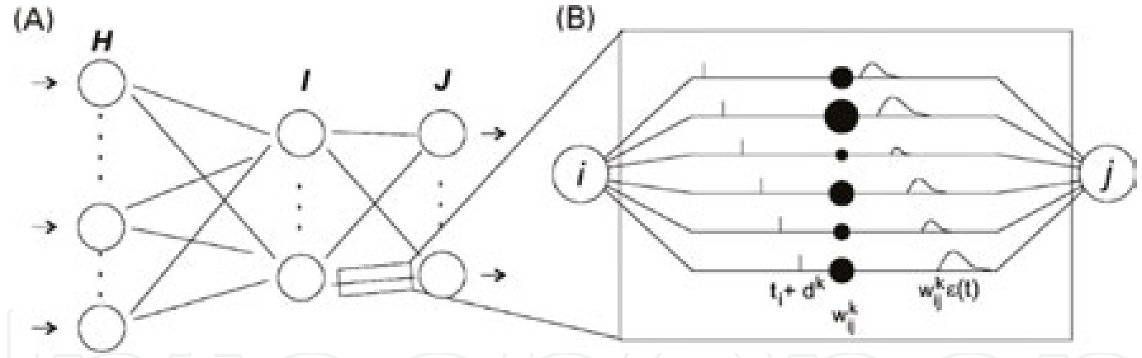

Figure 13. (A) Feed-forward SNNand (B) connection of multiple delay.

For each spiking neuron *j*, its internal state $x_j(t)$ is

$$x_j(t) = \sum_{i \in \Gamma_j} \sum_k w_{ij}^k \varepsilon(t - t_i - d_{ij}^k)$$

Neuron *i* is its predecessor and weight w_{ij}^k and delay d_{ij}^k are the synaptic connection between neuron *i* and *j*. t_i is the *i*th input from presynaptic neuron. If $x_j(t) \geq v$, v is pre-set threshold, the neuron is spiked and could transmit spike to other neurons. $\varepsilon(t)$is called spiking response function,

$$\varepsilon(t) = \begin{cases} \frac{t}{\tau} e^{1-\frac{t}{\tau}}, & t > 0 \\ 0, & t \le 0 \end{cases}$$

τ is the membrane potential decay time constant that determines the rise and decay time of the postsynaptic potential (PSP). **Figure 14** shows us how one spiking neuron begins to receive input pulse step by step and become spiked. Each neuron could begin to transmit information to its posterity only when this neuron is spiked. SNN implements temporal coding strategy which means the whole network is only based on one variable *t* which is initially set as 0. Then *t* increment 0.01 epoch by epoch until the output layer is spiked. The final output is the output layer spiking time *t*.

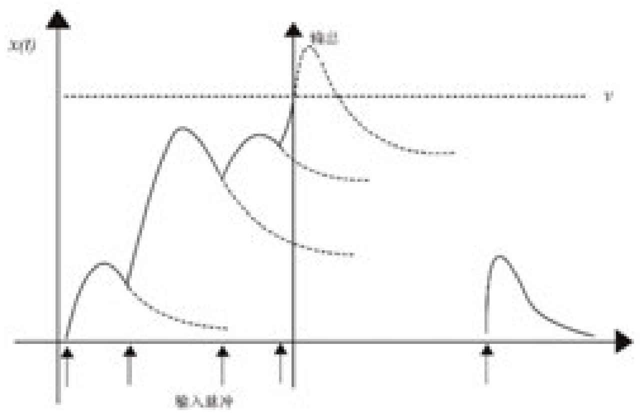

Figure 14. Process of neuron spike when receiving input spike.

Owing to the multiple delay connection, one SNN can achieve times of computation scale and ability than ANN in the same multilayer structure. With temporal coding, the whole network has only one variable *t*, thus the iterative process is simpler than ANN.

For supervised learning algorithm of SNN, Bohte et al. [21] first time give us a proved error-propagation supervised learning algorithm called spikeprop algorithm which is also based on GD. Adeli [22] revised the spikeprop algorithm and put forward another supervised learning algorithm quickprop.

3.4. Neural network encoding strategy

When dealing with different types of data for classification or prediction, encoding strategy similar to genetic coding needs being implemented to transform real problem to neural network format. In addition, because of the longtime of neural network training, some preprocessing method needs to be taken for extracting significant information or features thus accelerate neural network learning.

For instance, dealing with image classification, feature extraction method need being taken as preprocessing for neural network learning and the famous one is CNN. Furthermore, when using neural network for speech recognition, feature extraction is also a need for transforming continuous speech data into discrete neural network input.

4. Applying meta-heuristic algorithm on neural network training

The main idea of applying meta-heuristic on neural network training is using meta-heuristic algorithm instead of GD algorithm modifying weight training. That is from

$$\mathrm{j}w_{t+1} = w_t + GD \Rightarrow w_{t+1} = w_t + Meta$$

By using meta-heuristic algorithm's global optimum searching ability, researchers are aiming to train neural network faster than traditional GD algorithm. In the following part, Section 1 reviewed four researchers' work on implementing meta-heuristic on neural network training including GA on NN, PSO on NN and hybrid GA and PSO on NN.

4.1. GA on neural network

Leung et al. [10] first tried implementing GA on neural network training. Their work was published in IEEE Transactions on Neural Network, 2003.

An improved GA is put forward by Leung. Crossover operations, mutation operations and fitness function of GA are all redefined by Professor Leung. Firstly, when chromosomes p_1 and p_2 do crossover operation, four possible offsprings will be generated and one with the biggest fitness value will be chosen as offspring. The four possible crossover offsprings os_1 *to* os_4 are generated as regulations listed below:

$$os_c^1 = \left[os_1^1, os_2^1, \cdots os_n^1\right] = \frac{p_1 + p_2}{2}$$

$$os_c^2 = \left[os_1^2, os_2^2, \cdots os_n^2\right] = p_{\max}\left(1 - w\right) + \max(p_1, p_2)w$$

$$os_c^3 = \left[os_1^3, os_2^3, \cdots os_n^3\right] = p_{\min}\left(1 - w\right) + \min(p_1, p_2)w$$

$$os_c^4 = \left[os_1^4, os_2^4, \cdots os_n^4\right] = \frac{\left(p_{\min} + p_{\max}\right)\left(1 - w\right) + (p_1 + p_2)w}{2}$$

$p_{\max}=[para_{\max}^1, para_{\max}^1, \cdots, para_{\max}^n]$, $p_{\min}=[para_{\min}^1, para_{\min}^1, \cdots, para_{\min}^n]$ are calculated respectively. For instance, Max([1,−1,4], [−3,3,2]) = [1,3,4] and Min([1,−1,4],[−3,3,2]) = [−3,−1,2].

Secondly, mutation operations are redefined. The regulations are given below

$$os = \max(fitness\left(os_c^1\right), \mathrm{fitness}\left(os_c^2\right), \mathrm{fitness}\left(os_c^3\right), \mathrm{fitness}(os_c^4))$$

$$os' = os + [b_1 \Delta nos_1, b_2 \Delta nos_2, \cdots b_n \Delta nos_n]$$

os is the chromosome with biggest fitness value in all four possible offsprings. b_i random equals to 0 or 1 and Δnos_i is a random number making sure $para_{\min}^i \le os_i + b_i \Delta nos_i \le para_{\max}^i$. os' is the final generation after crossover operation and mutation operation.

Thirdly, the fitness value is defined. By adding parameters in the neural network mathematical expression, the actual output of GA-optimized neural network y_k equals to:

$$y_k = \sum_{j=1}^{n_h} \delta(s_{jk}^2) w_{jk} logsig\left[\sum_{i=1}^{n_i} \delta\left(s_{ij}^1\right) w_{ij} x_i - \delta\left(s_j^1 b_j^1\right)\right] - \delta\left(s_k^2\right) logsig\left(b_k^2\right)$$

in which $k = 1, 2, \cdots, n_{out}$, s_{ij} denotes link from ith neuron in input layer to jth neuron in hidden layer, s_{jk} denotes link from jth neuron in hidden layer to kth neuron in output layer, w_{jk} denotes weight between each neuron, b_k^1 *and* b_k^2 denote bias in input layer and hidden layer respectively, n_{in}, n_h *and* n_{out} denote the number of neurons of input layer, hidden layer and output layer, respectively.

The error of the whole network is defined as mean of all chromosomes:

$$\mathrm{err} = \sum_{k=1}^{n_{out}} \frac{\left|y_k - y_k^d\right|}{n_d}$$

in which n_d denotes the number of chromosomes used in the experiment, y_k^d denotes the desired output of output neuron k. Given the error of the network, GA is implemented to optimize the network, thus minimizing the error. The fitness function is defined as

$$\mathrm{fitness} = \frac{1}{1 + err}$$

the smaller the error and the bigger the fitness value. GA is implemented to find the global optimum of the fitness function, thus the parameter combinations of weight w are the trained weight for the network.

4.2. PSO on neural network

Gudise and Venayagamoorthy [5] implemented PSO on neural network training in 2003.

The fitness value of each particle (member) of the swarm is the value of the error function evaluated at the current position of the particle and position vector of the article corresponds to the weight matrix of the network.

Zhang et al. [7] developed a hybrid algorithm of BP and PSO that could balance training speed and accuracy.

The PSO algorithm was showed to converge rapidly during the initial stages of a global search, but around global optimum, the search process will become very slow. On the contrary, the gradient descending method can achieve faster convergent speed around global optimum, and at the same time, the convergent accuracy can be higher.

When the iteration process is approaching end and current best solution is near-global optimum, if the change of the weight in PSO is big, the result will vibrate severely. Under this condition, Zhang supposed with the increase of iteration time, the weight in PSO should decline with the iteration time's increasing to narrow the search range thus paying more attention to local search for global best. He suggests the weight decline linearly first, then decline nonlinearly as shown in **Figure 15**.

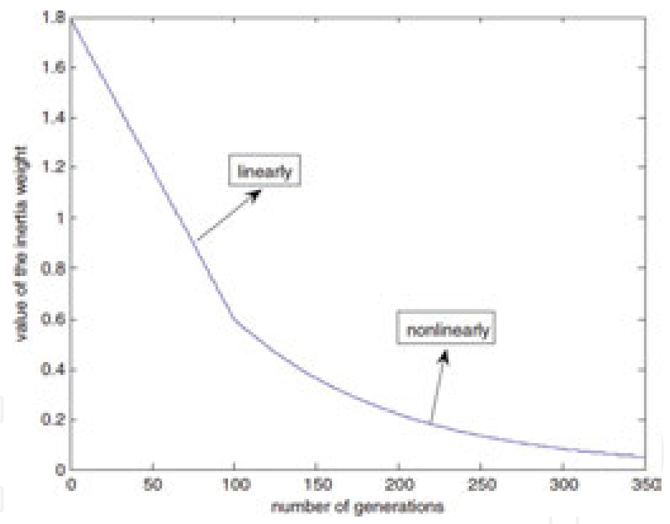

Figure 15. Change of weight in PSO with number of generations.

The concrete working process is summarized below:

For all particle p_i, it has a global best location $p_{globalbest}$. If the $p_{globalbest}$ keep unchanged for over 10 generations, that may infer the PSO pays too much time on global search thus BP is implemented for $p_{globalbest}$ to deep search for a better solution.

Similar to GA's implementation in neural network, the fitness function defined is also based on whole network's error and to minimize the error as the optimization of PSO.

The learning rate η of neural network is also controlled in the algorithm, as

$$\eta = k \times e^{-\eta_0 \cdot epoch}$$

where η is the learning rate, k and η_0 are constants, *epoch* is a variable that represents iterative times, through adjusting k and η_0, the reducing speed of learning rate is controlled.

By implementing the strategy that BP focusing on deep searching and PSO focusing on global searching, the hybrid algorithm has a very good performance.

4.3. Hybrid GA and PSO on neural network

Juang [6] hybrids GA and PSO thus optimize recurrent network's training. The work was published in IEEE Transactions on Systems, Man, and Cybernetics, 2004.

The hybrid algorithm called HGAPSO is put forward because the learning performance of GA may be unsatisfactory for complex problems. In addition, for the learning of recurrent network weights, many possible solutions exist. Two individuals with high fitness values are likely to have dissimilar set of weights, and the recombination may result in offspring with poor performance.

Juang put forward a conception of "elite" of the first half to enhance the next generation's performance. In each generation, after the fitness values of all the individuals in the same population are calculated, the top-half best-performing ones are marked. These individuals are regarded as elites.

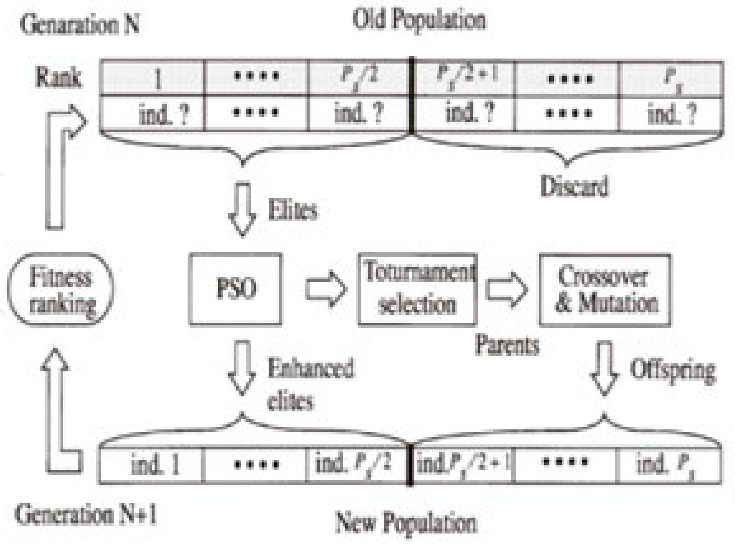

Figure 16. Working flow of HGAPSO.

In every epoch, the worse half of the chromosome is discarded. The better half is chosen for reproduction through PSO's enhancement. All elite chromosomes are regarded as particles in

PSO. By performing PSO on the elites, we may avoid the premature convergence in elite GAs and increase the search ability. Half of the population in the next generation is occupied by the enhanced individuals, the others by crossover operation. The working flow of algorithm is shown in **Figure 16**.

The crossover operation of HGAPSO is similar to normal GA, randomly selecting site on chromosome and exchange the sited piece of chromosome to finish the crossover operation. The crossover schematic diagram is shown in **Figure 17**. In HGAPSO, uniform mutation is adopted, that is, the mutated gene is drawn randomly, uniformly from the corresponding search interval.

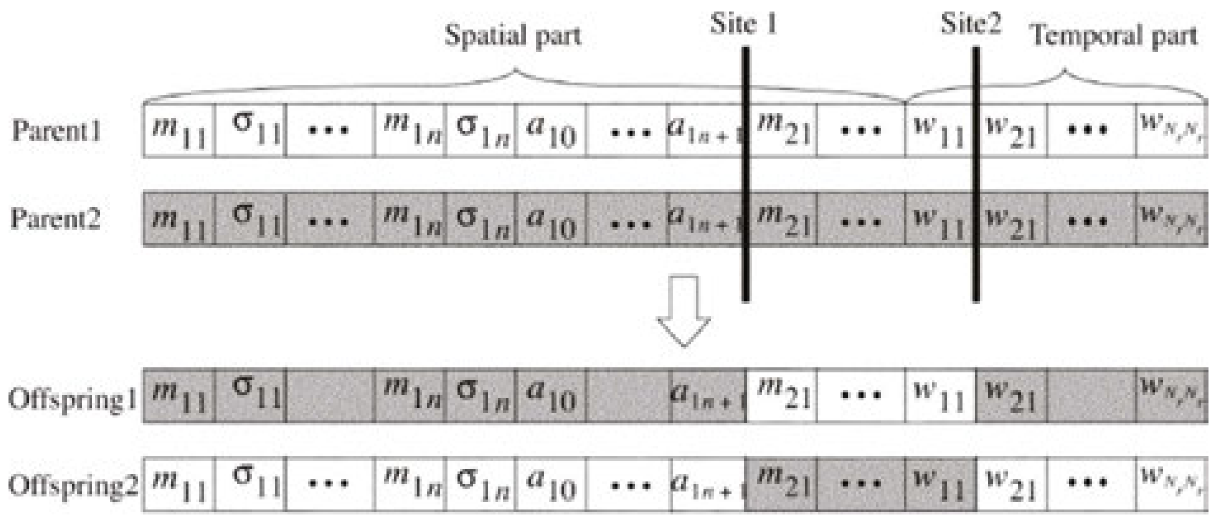
Spatial part
Site 1
Site2
Temporal part
Parent1
Parent2
Offspring1
Offspring2
m_{11} σ_{11} ... m_{1n} σ_{1n} a_{10} ... a_{1n+1} m_{21} ... w_{11} w_{21} ... $w_{N_rN_r}$

Figure 17. Schematic diagram of crossover operation.

5. DL and RBM

When dealing with image classification or other problems, traditional method is using preprocessing transforming data as input values for neural network learning while using DL method for classification, raw data (pixel values) are used as input values. This will keep to the maximum extent protecting all information regardless of useful or not from being destroyed by extraction methods. The most advantage lies that all the extraction methods are based on expert knowledge and expert choice and thus are not extensible to other problems, while DL algorithm can overcome these limitations by using all data with its powerful processing ability. A CNN is shown in **Figure 18**.

The Boltzmann machine only has two layers (as **Figure 19**), the first layer is the input layer and the second layer is the output layer, although the structure is very simple and only contains two layers, its mathematical function in it is not simple. Herein we need to introduce the following probability equation to know the RBM.

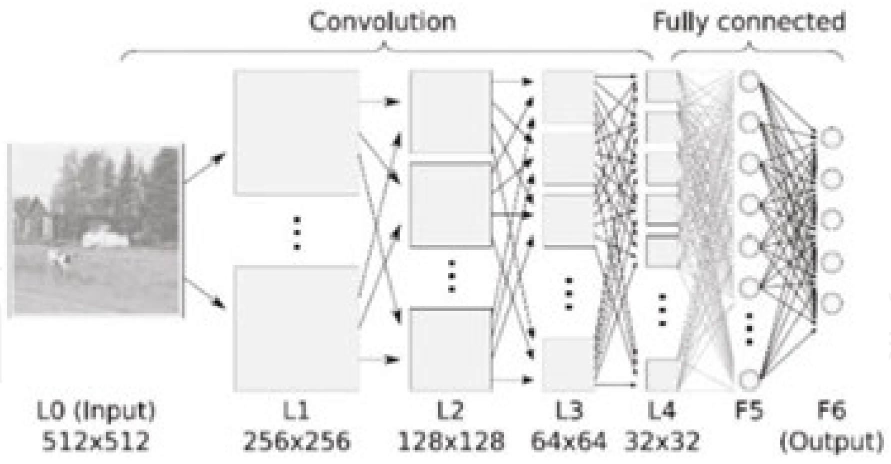

Figure 18. Convolutional neural network.

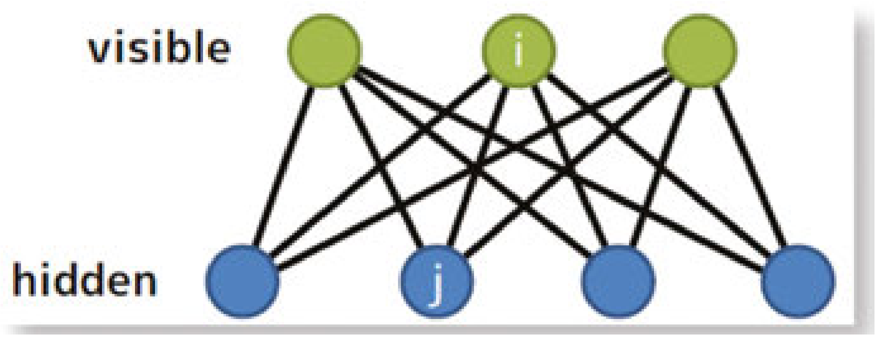

Figure 19. Restricted Boltzmann machine.

From the equation we know there are three items in it, they are the energy of the visible layer neural, the energy of the hidden layer neural, and the energy is consisted of the two layers. From the energy function of the RBM, we can see there are three kinds of parameters lie in the RBM, different from the neural network, each neuron has a parameter too, but for the neural network only the connection of two layers has the parameters. Also, the neural network does not carry any energy function. They use the exponential function to express the potential function. There are also the probabilities existing in the RBM. They work exactly in the same way as the two kind of the probabilities such as $p(v|h)$ and $p(h|v)$. It has a similarity with probability graph model in details, examples are Bayesian network and Markov network. It looks like a Bayesian network because of the conditional probability. On the other hand it does not look like a Bayesian because of the two direction probabilities. These probabilities are lying on the two variables that have only one direction.

Compared with the Markov network, the RBM seems to be having a little bit relation with it, because the RBM has the energy function just as the Markov network. But it is not so much alike because the RBM's variable has parameter, and the Markov network does not have any parameter. Furthermore, the Markov network does not have conditional probability because it has no direction but just the interaction. From the graph's perspective, variables in the

Markov network use cliques or clusters to represent the relations of close and communicated variables. It uses the production of the potentials of the clique to express the joint probability instead of conditional probability just like the RBM. Its input data are the kind of the Boolean data, within the range between 0 and 1.

The training way of the RBM is to maximize the probability of the visible layer *p*(*v*), and to generate the distribution of the input data. RBM is a kind of special Markov random function and a special kind of Boltzmann machine. Its graphical model is corresponding to the factor product analysis.

Different from the probability graphical model, the RBM's joint distribution directly uses the energy function of both visible layer *v* and hidden layer *h* to define instead of potential of it given as

$$E\left(v,h\right) = -a^T v - b^T h - v^T Wh$$

$$P\left(v,h\right) = \frac{1}{Z} e^{-E\left(v,h\right)}$$

Later we will know that *Z* is the partition function defined as the sum of $e^{-E}(V,h)$ over all the possible configurations. In other words, it is just a constant normalizing the sum over all the possible hidden layer configurations.

$$P\left(v\right) = \frac{1}{Z} \sum_h e^{-E\left(v,h\right)}$$

The hidden unit activations are mutually independent given the activations. That is, for *m* visible and *n* hidden units, the conditional probability of a configuration of the visible unit *v*, given a configuration

$$P\left(v|h\right) = \prod_{i=1}^{m} P\left(v_i|h\right)$$

Conversely, the conditional probability of *h* given *v* is $P(h \mid v) = \prod_{n}^{j=1} P(h_j \mid v)$. Our goal is to infer the weights that maximize the marginal of the visible, in details we can step through the following equation to infer and learn the RBM.

$$\arg\max_w E\left[\sum_{v \in V} \log P(v)\right]$$

As for the training algorithm, the main idea is also applied GD idea into RBM. Hinton put forward CD [23] as a faster learning algorithm. Firstly, the derivative of the log probability of a training vector with respect to a weight is computed as

$$\frac{\partial logP(v)}{\partial w_{ij}} = \left\langle v_i h_j \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{model}$$

where the angle brackets are used to denote expectations under the distribution specified by the subscript that follows. This leads to a very simple learning rule for performing stochastic steepest ascent in the log probability of the training data:

$$\Delta w_{ij} = \varepsilon \left(\left\langle v_i h_j \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{model} \right)$$

where ε is a learning rate.

Because there are no direct connections between hidden units in an RBM, it is very easy to get an unbiased sample of $v_j h_{jdata}$. Given a randomly selected training image, v, the binary state, h_j, of each hidden unit, j, is set to 1 with probability

$$p\left(h_j = 1 \middle| v\right) = \sigma(b_j + \sum_i v_i w_{ij})$$

where b_j is the current state of hidden neuron j, $\sigma(x)$ is the logistic sigmoid function $\sigma(x) = 1/(1+\exp(-x))$. $v_j h_j$ is then an unbiased sample. The CD is used to calculate the latter part $v_j h_{jmodel}$. Details can be found in their respective publications.

Considering the complicated computation of implementing CD, the training process of RBM is not easy. Under this condition, implementing meta-heuristic on RBM training to substitute CD is of high possibility.

6. Conclusion and discussion

Meta-heuristic has successfully implemented in neural network training. The algorithm used includes GA, PSO, their hybrid and many other meta-heuristic algorithms. Moreover, feed-forward BP neural network and SNN [24] are all trained with tests on famous classification problems.

The basic structure of DL is similar to traditional neural network. CNN is a special neural network with different weight computation regulations and RBM is a weighted biodimensional neural network or biodimensional graph. Their training processes are also mainly executed through iterative formula on error which is similar to traditional neural network's training.

Having seen the above two concluding remarks, it can be assumed that it is of high possibility of applying meta-heuristic in DL to speed up training without declining performance. However, relevant publications along this direction are still rare.

Lastly, there still exists a question that goes under today's computation ability, especially the GPU whose computation ability is several times stronger than CPU that has been widely used in industrial area. One may ask, although elegant, is meta-heuristic still necessary? This question is not easy to answer. However, one can be certain that when traversing all the possible solutions is certainly highly time-consuming. Search for near-optimal results by meta-heuristic is still useful that can provide us a reasonable searched result (instead of non optimal or far from optimal) near a global optimum value at an acceptable computational cost.

Author details

Zhonghuan Tian and Simon Fong*

*Address all correspondence to: ccfong@umac.mo

Department of Computer and Information Science, University of Macau, Macau SAR, China

References

[1] Hinton, G. E., Osindero, S., and Teh, Y. W. A fast learning algorithm for deep belief nets. Neural Computation. 2006;18(7):1527-1554.

[2] Yu Kai, Jia Lei, Chen Yuqiang, and Xu Wei. Deep learning: yesterday, today, and tomorrow. Journal of Computer Research and Development. 2013;50(9):1799-1804.

[3] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.

[4] Izadinia, Hamid, Bryan C. Russell, Ali Farhadi, Matthew D. Hoffman, and Aaron Hertzmann. "Deep classifiers from image tags in the wild." In*Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions*, pp. 13–18. ACM, 2015.

[5] Gudise, V. G. and Venayagamoorthy, G. K. Comparison of particle swarm optimization and back propagation as training algorithms for neural networks. In: Proceedings of In Swarm Intelligence Symposium SIS'03; 2006. p. 110-117.

[6] Beheshti, Z. and Shamsuddin, S. M. H. A review of population-based meta-heuristic algorithms. International Journal of Advances in Soft Computing & Its Applications 2013;5(1):1-35.

[7] Zhang, J. R., Zhang, J., Lok, T. M., and Lyu, M. R. A hybrid particle swarm optimization–back-propagation algorithm for feed forward neural network training. Applied Mathematics and Computation. 2007;185(2):1026-1037.

[8] Juang, C. F. A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions 2004;34(2):997-1006.

[9] Meissner, M., Schmuker, M., and Schneider, G. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. BMC Bioinformatics. 2006;7(1):125.

[10] Leung, F. H., Lam, H. K., Ling, S. H., and Tam, P. K. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Transactions on Neural Networks. 2003;14(1):79-88.

[11] Goldberg, D. E. and Holland, J. H. Genetic algorithms and machine learning. Machine Learning. 1988;3(2):95-99.

[12] Kennedy, J. Particle Swarm Optimization; Springer, USA; 2010. p. 760-766

[13] Glover, F. Tabu search-part I. ORSA Journal on Computing. 1989;1(3):190-206.

[14] Wang, K. P., Huang, L., Zhou, C. G., and Pang, W. Particle swarm optimization for traveling salesman problem. In: International Conference on Machine Learning and Cybernetics; IEEE; 2003. p. 1583-1585.

[15] Clerc, M. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: New Optimization Techniques in Engineering; Springer, Berlin Heidelberg; 2004. p. 219-239.

[16] Cochocki, A. and Unbehauen, R. Neural Networks for Optimization and Signal Processing; John Wiley & Sons, Inc., New York, NY, USA; 1993.

[17] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review. 1958;65(6):386.

[18] Nitta, T. Solving the XOR problem and the detection of symmetry using a single complex-valued neuron. Neural Networks. 2003;16(8):1101-1105.

[19] Maass, W. Networks of spiking neurons: The third generation of neural network models. Neural Networks. 1997;10(9):1659-1671.

[20] Kistler, W. M., Gerstner, W., and Hemmen, J. Reduction of the Hodgkin-Huxley equations to a single-variable threshold model. Neural Computation. 1997;9(5):1015-1045.

[21] Bohte, S. M., Kok, J. N., and La Poutre, H. Error-back propagation in temporally encoded networks of spiking neurons. Neurocomputing. 2002;48(1):17-37.

[22] Ghosh-Dastidar, S. and Adeli, H. Improved spiking neural networks for EEG classification and epilepsy and seizure detection. Integrated Computer-Aided Engineering. 2007;14(3):187-212.

[23] Hinton, G. A practical guide to training restricted Boltzmann machines. Momentum. 2010;9(1):926.

[24] Pavlidis, N. G., Tasoulis, D. K., Plagianakos, V. P., Nikiforidis, G., and Vrahatis, M. N. Spiking neural network training using evolutionary algorithms. In: Proceedings of IEEE International Joint Conference on Neural Networks IJCNN'05; July 31–August 2005; IEEE; 2005. p. 2190-2194.

[25] Schmidhuber, J. Deep learning in neural networks: An overview. Neural Networks 2015;61:85-117.